

# Funneled Bayesian Optimization for Design, Tuning and Control of Autonomous Systems

Ruben Martinez-Cantin  
Centro Universitario de la Defensa, Zaragoza, Spain  
rmcantin@unizar.es

## Abstract

Bayesian optimization has become a fundamental global optimization algorithm in many problems where sample efficiency is of paramount importance. Recently, there has been proposed a large number of new applications in fields such as robotics, machine learning, experimental design, simulation, etc. In this paper, we focus on several problems that appear in robotics and autonomous systems: algorithm tuning, automatic control and intelligent design. All those problems can be mapped to global optimization problems. However, they become hard optimization problems. Bayesian optimization internally uses a probabilistic surrogate model to learn from the process and reduce the number of samples required. The *de facto* standard surrogate model is a Gaussian process due to its flexibility to represent a distribution over functions. In order to generalize to unknown functions in a black-box fashion, the common assumption is that the underlying function can be modeled with a stationary process. Nonstationary Gaussian process regression cannot generalize easily and it typically requires prior knowledge of the function. Some works have designed techniques to generalize Bayesian optimization to nonstationary functions in an indirect way, but using techniques originally designed for regression, where the objective is to improve the quality of the surrogate model everywhere. Instead optimization should focus on improving the surrogate model near the optimum. In this paper, we present a novel kernel function specially designed for Bayesian optimization, that allows nonstationary behavior of the surrogate model in an adaptive local region. In our experiments, we found that this new kernel results in an improved local search (exploitation), without penalizing the global search (exploration). We provide extensive results in well-known benchmarks and real applications to show that the new method is able to outperform the state of the art in Bayesian optimization both in stationary and nonstationary problems.

## 1 Introduction

Many problems in engineering, computer science, economics, etc., require to find the extremum of an unknown real valued function using as few evaluations as possible. In many cases, those functions represent actual industrial processes, expensive trials or time consuming computations or simulations. The optimization process must consider the actual budget and limitations of gathering new evaluations. Then, sample efficiency becomes the key element. Furthermore, those functions might be highly multimodal, requiring a global solution.

Bayesian optimization, also found in the literature with the names of Bayesian Sampling [56], Efficient Global Optimization (EGO) [31], Sequential Kriging Optimization (SKO) [27], Sequential Model-Based Optimization (SMBO) [29] or Bayesian guided pattern search [59], is a classic optimization method [37, 43] which has become quite popular recently for being a sample efficient method of global optimization [31]. It has been applied with great success to autonomous algorithm tuning [39], specially for machine learning applications [52, 16], robot planning [41], control [10], task optimization [36], reinforcement learning [42, 13], structural design [17], sensor networks [55, 20], simulation design [7], circuit design [59], ecology [63], biochemistry [12], dynamical modeling of biological systems [61], etc. Recent works have found connections between Bayesian optimization and the way biological systems adapt and search in nature, such as human active search [5] or animal adaptation to injuries [11].

Bayesian optimization uses a Bayesian surrogate model, that is, a distribution over target functions  $P(f)$  to incorporate all available information during the optimization procedure. This distribution can

capture both the prior information available and all the information from each observation. Therefore, the surrogate model provides a memory [45] that improves the sample efficiency of the method by considering the whole history of trials and evaluations during the optimization procedure. Besides, this model is sample efficient by using two principles.

First, it can be updated recursively as outcomes are available from the evaluated trials  $y_i = f(\mathbf{x}_i)$

$$P(f|\mathbf{X}_{1:i}, \mathbf{y}_{1:i}) = \frac{P(\mathbf{x}_i, y_i|f)P(f|\mathbf{X}_{1:i-1}, \mathbf{y}_{1:i-1})}{P(\mathbf{x}_i, y_i)}, \quad (1)$$

$\forall i = 2 \dots n$  where  $\mathbf{X}_{1:i}$  is a matrix with all the inputs  $\{\mathbf{x}_j\}_{j=1}^i$  and  $\mathbf{y}_{1:i}$  is a vector with all the outcomes  $\{y_j\}_{j=1}^i$ . By using this method, the information is always updated<sup>1</sup>.

Second, it allows to compute the optimal decision/action  $\mathbf{a}$  of selecting the next trial  $\mathbf{x}_{n+1}$  by maximizing (minimizing) the expected utility (loss):

$$\mathbf{a}^{BO} = \arg \min_{\mathbf{a}} \int \delta_n(f, \mathbf{a}) dP(f|\mathbf{x}_{1:n}, \mathbf{y}_{1:n}) \quad (2)$$

where  $\delta_n(f, \mathbf{a})$  is the *optimality criterion* or *regret function* that drives the optimization towards the optimum  $\mathbf{x}^*$ . For example, we can use the *optimality gap*  $\delta_n(f, \mathbf{a}) = f(\mathbf{x}_n) - f(\mathbf{x}^*)$  to get the optimal outcome, the *Euclidean distance error*  $\delta_n(f, \mathbf{a}) = \|\mathbf{x}_n - \mathbf{x}^*\|_2$  to get the optimal input, or the *relative entropy*  $\delta_n(f, \mathbf{a}) = H(\mathbf{x}^*|\mathbf{x}_{1:n-1}) - H(\mathbf{x}^*|\mathbf{x}_{1:n})$  to maximize the information about the optimum.

In summary, Bayesian optimization is the combination of two main components: a surrogate model which captures all prior and observed information and a decision process which performs the optimal action, i.e.: where to sample next, based on the previous model. This methodology in two steps of modeling and decision is connected to many fields. In the way points are selected, the optimization problem can be considered an active learning problem on the estimation of the optimum [25, 14]. Other authors have drawn connections between Bayesian optimization and some reinforcement learning setups such as multi-armed bandits [55], POMDPs [60] or even active reinforcement learning [42]. These two components also hide extra computational cost. On one hand, we need to update the surrogate model and, on the other hand, optimize the criterion function. However, this additional cost can be compensated by the reduced number of target function evaluations thanks to the sample efficiency of the method. Therefore, Bayesian optimization is specially suitable for expensive black-box functions and trial-and-error methodologies.

Under these conditions, the quality of the surrogate model is of paramount importance as it also affects to the optimality of the decision process. Earliest versions of Bayesian optimization used Wiener processes [37] or Wiener fields [44] as surrogate models. Similar methods used radial basis functions [24] or branch and bound with polynomials [51]. It was the seminal paper of Jones et al. [31] that introduced the use of Gaussian processes, also called Kriging, as a Bayesian surrogate function. Jones also wrote an excellent review on this kind of surrogate models [30]. Recently, other Bayesian models have become popular like Student's t processes [50, 40], treed Gaussian processes [59, 3], random forests [29], tree-structured Parzen estimators [4] or deep neural networks [53]. In the case of discrete inputs, the Beta-Bernoulli bandit setting provides an equivalent framework [48].

However, the Gaussian process (GP) is still the most popular model due to its accuracy, robustness and flexibility, because Bayesian optimization is mainly used in black or grey-box scenarios. The range of applicability of a Gaussian process is defined by its kernel function, which sets the family of functions that is able to represent through the reproducing kernel Hilbert space (RKHS) [47]. In fact, regret bounds for Bayesian optimization using Gaussian processes are always defined in terms of specific kernel functions [14, 55, 9]. From a practical point of view, the standard procedure is to select a generic kernel function, such as the Gaussian (square exponential) or Matérn kernels, and estimate the kernel hyperparameters from data. One property of these kernels is that they are stationary. Although it might be a reasonable assumption in a black box setup, we show in Section 3 that this reduces the efficiency of Bayesian optimization in most situations. It also limits the potential range of applications. Furthermore, nonstationary methods usually require extra knowledge of the function (e.g.: the global trend or the space partition). Being global properties, gathering this knowledge from data requires global sampling, which is contrary to the Bayesian optimization methodology.

<sup>1</sup>For the remainder of the paper we use the colon notation  $\mathbf{v}_{1:T}$  to represent a stacked vector  $\mathbf{v}_{1:T} = [v_1, \dots, v_T]^T$  or a stacked matrix  $\mathbf{V}_{1:T} = [\mathbf{v}_1, \dots, \mathbf{v}_T]$ .

The main contribution of the paper is a new set of adaptive kernels for Gaussian processes that are specifically designed to model functions from nonstationary processes but focused on the region near the optimum. Thus, the new model maintains the global exploration/local exploitation trade off. This idea results in an improved efficiency and applicability of any Bayesian optimization based on Gaussian processes. We call this new method *Spartan Bayesian Optimization* (SBO). The algorithm has been extensively evaluated in many scenarios and applications. Besides some standard optimization benchmarks, the method has been evaluated in automatic algorithm tuning for machine learning applications, optimal policy learning in reinforcement learning scenarios and autonomous wing design of an airplane using realistic CFD simulations. In our results, we have found that SBO reaches its best performance in problems that are clearly nonstationary, where the local and global shape of the function are different. However, our tests have also shown that SBO can improve the results of Bayesian optimization in all those scenarios. From an optimization point of view, these new kernels result in an improved local search while maintaining global exploration capabilities, similar to other locally-biased global optimization algorithms [18].

The algorithm is called *Spartan* as it follows the same intuition and analogous strategy as the Greek forces at the end of the *Battle of Thermopylae*. The most likely theory claims that, the last day of the battle, a small group of forces led by spartan King Leonidas stood in the narrow pass of the Thermopylae to block the Persian cavalry, while the rest of the forces retreated to cover more terrain and avoid being surrounded by the Persian moving through a mountain path [38]. This dual strategy of allocate global resources sparsely while maintaining a local dense vanguard at a strategic location is emphasized within *Spartan Bayesian Optimization*.

## 2 Bayesian optimization with Gaussian processes

We start describing the ingredients for a Bayesian optimization algorithm using Gaussian processes as surrogate model. Consider the problem of finding the minimum of an unknown real valued function  $f : \mathbb{X} \rightarrow \mathbb{R}$ , where  $\mathbb{X}$  is a compact space,  $\mathbb{X} \subset \mathbb{R}^d, d \geq 1$ . In order to find the minimum, the algorithm has a maximum budget of  $N$  evaluations of the target function  $f$ . The purpose of the algorithm is to select the best query points at each iteration such as the optimization gap or regret is minimum for the available budget.

Without loss of generality, for the remainder of the paper we are going to assume that the surrogate model  $P(f)$  is a Gaussian process  $\mathcal{GP}(\mathbf{x}|\mu, \sigma^2, \boldsymbol{\theta})$  with inputs  $\mathbf{x} \in \mathbb{X}$ , scalar outputs  $y \in \mathbb{R}$  and an associate kernel or covariance function  $k(\cdot, \cdot)$  with hyperparameters  $\boldsymbol{\theta}$ . The hyperparameters are estimated using Monte Carlo Markov Chain (MCMC) resulting in  $m$  samples  $\boldsymbol{\Theta} = \{\boldsymbol{\theta}_i\}_{i=1}^m$ . Concretely, we use the slice sampling algorithm which has already been used in Bayesian optimization [52], although bootstrapping could equally be used [33]. One advantage of using Gaussian processes as a distributions over functions is that new observations of the target function  $(x_i, y_i)$  can be easily used to update the distribution over functions. Furthermore, the posterior distribution is also a GP. Therefore, the posterior can be used as a prior for the next iteration in a recursive algorithm. In fact, being a non-parametric method, the algorithm of adding a new sample can be highly optimized provided that we do not update the hyperparameters every iteration [40].

Given a dataset at step  $n$  of query points  $\mathbf{X} = \{\mathbf{x}_{1:n}\}$  and its respective outcomes  $\mathbf{y} = \{y_{1:n}\}$ , then the prediction of the Gaussian process at a new query point  $\mathbf{x}_q$ , with kernel  $k_i$  conditioned on the  $i$ -th hyperparameter sample  $k_i = k(\cdot, \cdot | \boldsymbol{\theta}_i)$  is a normal distribution such as  $y_q \sim \sum_{i=1}^m \mathcal{N}(\mu_i, \sigma_i^2 | \mathbf{x}_q)$  where:

$$\begin{aligned} \mu_i(\mathbf{x}_q) &= \mathbf{k}_i(\mathbf{x}_q, \mathbf{X}) \mathbf{K}_i(\mathbf{X}, \mathbf{X})^{-1} \mathbf{y} \\ \sigma_i^2(\mathbf{x}_q) &= k_i(\mathbf{x}_q, \mathbf{x}_q) - \mathbf{k}_i(\mathbf{x}_q, \mathbf{X}) \mathbf{K}_i(\mathbf{X}, \mathbf{X})^{-1} \mathbf{k}_i(\mathbf{X}, \mathbf{x}_q) \end{aligned} \quad (3)$$

being  $\mathbf{k}_i(\mathbf{x}_q, \mathbf{X})$  the corresponding cross-correlation vector of the query point  $\mathbf{x}_q$  with respect to the dataset  $\mathbf{X}$

$$\mathbf{k}_i(\mathbf{x}_q, \mathbf{X}) = [k_i(\mathbf{x}_q, \mathbf{x}_1), \dots, k_i(\mathbf{x}_q, \mathbf{x}_n)]^T$$

and  $\mathbf{K}_i(\mathbf{X}, \mathbf{X})$  is the Gram matrix corresponding to kernel  $k_i$  for the dataset  $\mathbf{X}$

$$\mathbf{K}_i(\mathbf{X}, \mathbf{X}) = \begin{pmatrix} k_i(\mathbf{x}_1, \mathbf{x}_1) & \dots & k_i(\mathbf{x}_1, \mathbf{x}_n) \\ \vdots & \ddots & \vdots \\ k_i(\mathbf{x}_n, \mathbf{x}_1) & \dots & k_i(\mathbf{x}_n, \mathbf{x}_n) \end{pmatrix} + \sigma_n^2 \mathbf{I}$$

where  $\sigma_n^2$  is a noise or nugget term to represent stochastic functions [27] or surrogate missmodeling [23]. Note that, because we use a sampling distribution of  $\theta$  the predictive distribution at any point  $\mathbf{x}$  is a mixture of Gaussians.

The problem with the regret functions that we presented for equation (2) is that all of them rely on the knowledge of the optimum  $\mathbf{x}^*$ . Thus, we cannot use any of those directly. Instead, the Bayesian optimization literature have developed proxies of those functions, called *acquisition functions*. To select the next point at each iteration, we use the *expected improvement* criterion [43] as a proxy of the optimality gap criterion. The expected improvement is defined as the expectation of the improvement function  $I(\mathbf{x}) = \max(0, \rho - f(\mathbf{x}))$ . The improvement is defined over a incumbent target  $\rho$ , which in many applications is considered to be the *best outcome* until that iteration  $\rho = y_{best}$ . Other incumbent values could be considered, specially in the presence of noise. Taking the expectation over the mixture of Gaussians of the predictive distribution, we can compute the expected improvement as:

$$\begin{aligned} EI(\mathbf{x}) &= \mathbb{E}_{p(y|\mathbf{x},\theta)} [\max(0, \rho - f(\mathbf{x}))] \\ &= \sum_{i=1}^m [(\rho - \mu_i) \Phi(z_i) + \sigma_i \phi(z_i)] \end{aligned} \quad (4)$$

where  $\phi$  and  $\Phi$  are the corresponding Gaussian probability density function (PDF) and cumulative density function (CDF), being  $z_i = (\rho - \mu_i)/\sigma_i$ . In this case,  $(\mu_i, \sigma_i^2)$  is the prediction computed with Equation (3).

Assuming that the incumbent target is the best observation up to iteration  $n - 1$ , then at iteration  $n$ , we select the next query at the point that maximizes the corresponding expected improvement:

$$\mathbf{x}_n = \arg \max_{\mathbf{x}} \sum_{i=1}^m [(y_{best} - \mu_i) \Phi(z_i) + \sigma_i \phi(z_i)] \quad (5)$$

Finally, in order to avoid bias and guarantee global optimality, we rely on an initial design of  $p$  points based on *Latin Hypercube Sampling* (LHS) following the recommendation in the literature [31, 9, 32]. Algorithm 1 summarizes the basic steps in Bayesian optimization.

---

**Algorithm 1** Bayesian optimization (BO)

---

- 1: Initial design of  $p$  points using any prior information (e.g.: LHS to maximize coverage)
  - 2:  $\mathbf{X} \leftarrow \mathbf{x}_{1:p}$     $\mathbf{y} \leftarrow y_{1:p}$
  - 3: **for**  $n = p \dots N$  **do** ▷ Available budget of  $N$  queries
  - 4:    **for**  $i = 1 \dots m$  **do** ▷ Iterate over MCMC samples
  - 5:      $\mu_i(\mathbf{x}) \leftarrow \mathbf{k}(\mathbf{x}, \mathbf{X}|\theta_i) \mathbf{K}(\mathbf{X}, \mathbf{X}|\theta_i)^{-1} \mathbf{y}$  ▷ Predicted mean
  - 6:      $\sigma_i^2(\mathbf{x}) \leftarrow k(\mathbf{x}, \mathbf{x}|\theta_i) - \mathbf{k}(\mathbf{x}, \mathbf{X}|\theta_i) \mathbf{K}(\mathbf{X}, \mathbf{X}|\theta_i)^{-1} \mathbf{k}(\mathbf{X}, \mathbf{x}|\theta_i)$  ▷ Predicted variance
  - 7:    **end for**
  - 8:     $\Theta \leftarrow \{\theta_i\}_{i=1}^m$  ▷ Update the hyperparameters using MCMC
  - 9:     $\mathbf{x}_n = \arg \max_{\mathbf{x}} EI(\mathbf{x})$  ▷ Expected improvement, see (5)
  - 10:     $y_n \leftarrow f(\mathbf{x}_n)$      $\mathbf{X} \leftarrow add(\mathbf{x}_n)$      $\mathbf{y} \leftarrow add(y_n)$
  - 11: **end for**
- 

### 3 Nonstationary Gaussian processes

Many applications of Gaussian process regression, including Bayesian optimization, are based on the assumption that the process is stationary. This is a reasonable assumption for black-box optimization as it does not assume any extra information on the evolution of the function in the space. For example, the use of the squared exponential (SE) kernel in GPs is quite frequent. Another popular kernel in GP regression is the Matérn kernel family:

$$k_{SE}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}r^2\right) \quad (6)$$

$$k_{Matern}(\mathbf{x}, \mathbf{x}') = \frac{2^{1-\nu}}{\Gamma(\nu)} \left(\sqrt{2\nu}r\right)^\nu K_\nu\left(\sqrt{2\nu}r\right) \quad (7)$$

where  $r^2 = (\mathbf{x} - \mathbf{x}')^T \Lambda (\mathbf{x} - \mathbf{x}')$  with  $\Lambda = \theta_l^{-1} \mathbf{I}$ . For these kernels,  $\theta_l$  represents the length-scale hyperparameter that captures the smoothness or variability of the function. In the Matérn kernel,  $K_\nu$  is a modified Bessel function. The Matérn kernel is usually computed for values of  $\nu$  that are half-integers  $\nu = p + 1/2$  where  $p$  is a non-negative integer, because the function becomes simpler.

$$k_{M,\nu=1/2}(\mathbf{x}, \mathbf{x}') = \exp(-r) \quad (8)$$

$$k_{M,\nu=3/2}(\mathbf{x}, \mathbf{x}') = \exp(-\sqrt{3}r) \left(1 + \sqrt{3}r\right) \quad (9)$$

$$k_{M,\nu=5/2}(\mathbf{x}, \mathbf{x}') = \exp(-\sqrt{5}r) \left(1 + \sqrt{5}r + \frac{5}{3}r^2\right) \quad (10)$$

The value of  $\nu$  is related to the smoothness of the functions as it directly affects the  $k$ th-differentiability of the underlying process. The process  $g(x)$  defined by the Matérn kernel  $k(\cdot, \cdot)$  is  $k$ -times mean square differentiable if and only if  $k > \nu$  [47]. Furthermore, for  $\nu \rightarrow \infty$ , we obtain the SE kernel from equation (6).

The analysis of the kernel length-scale is very important for optimization. Small values of  $\theta_l$  will be more suitable to capture signals with high frequency components; while large values of  $\theta_l$  result in a model for low frequency signals or flat functions. This property also holds for kernels with automatic relevance determination (ARD) [47], where  $\theta_l$  becomes a vector with a length-scale parameter per dimension. In Bayesian optimization, the Matérn kernel with  $\nu = 5/2$  and ARD is frequently used for its performance in many benchmarks.

This length-scale estimation results in an interesting behavior in Bayesian optimization. For the same distance between points, a kernel with smaller length-scale will result in higher predictive variance, therefore the exploration will be more aggressive. This idea was previously explored in Wang et al. [62] by forcing smaller scale parameters to improve the exploration. More formally, in order to achieve no-regret convergence to the minimum, the target function must be an element of the reproducing kernel Hilbert space (RKHS) characterized by the kernel  $k(\cdot, \cdot)$  [9, 55]. For a set of kernels like the SE or Matérn, it can be shown that:

**Proposition 1.** *Given two kernels  $k_l$  and  $k_s$  with large and small length scale hyperparameters respectively, any function  $f$  in the RKHS characterized by a kernel  $k_l$  is also an element of the RKHS characterized by  $k_s$  [62].*

Thus, using  $k_s$  instead of  $k_l$  is safer in terms of guaranteeing convergence. However, if the small kernel is used everywhere, it might result in unnecessary sampling of smooth areas.

**Definition 1.** Let  $f : \mathbb{R}^d \rightarrow \mathbb{R}$  be a function and  $\mathcal{H}_k$  be the reproducing kernel Hilbert space generated by kernel  $k(\cdot, \cdot)$ .

- We say that a function  $f(\mathbf{x})$  is *stationary* if  $\exists k(\mathbf{x}, \mathbf{x}') = k(\tau)$  where  $\tau = \mathbf{x} - \mathbf{x}'$  and  $f \in \mathcal{H}_k$ .
- In contrast, we say that a function  $f(\mathbf{x})$  is *nonstationary* if  $\nexists k(\mathbf{x}, \mathbf{x}') = k(\tau)$  where  $\tau = \mathbf{x} - \mathbf{x}'$  and  $f \in \mathcal{H}_k$ .
- Finally, we say that a function  $f(\mathbf{x})$  is *local stationary* if there is a subset  $\mathcal{X} \subset \mathbb{R}^d$  so that the function is stationary  $\forall \mathbf{x} \in \mathcal{X}$  and nonstationary  $\forall \mathbf{x} \in \mathbb{R}^d \setminus \mathcal{X}$ .

According to the previous definition, most applications of Bayesian optimization are nonstationary or local stationary. Take for example the reinforcement learning problems of Section 5.4. In that scenario, an agent  $A$  is trying to find the optimal policy  $\pi^*$  which produce the behavior that maximize a reward function  $R$ :

**Example 1.** Let us consider a biped robot (agent) trying to get the walking pattern (policy) that maximizes the walking speed (reward). In this setup, there are some policies that reach undesirable states or result in a failure condition, like the robot falling or losing the upright posture. Then, the system returns a null reward or arbitrary penalty. In cases where finding a stable policy is difficult, the reward function may end up being almost flat, except for a small region of successful policies where the reward is actually informative in order to maximize the speed.

Modeling these kind of functions with Gaussian processes require kernels with different length scales for the flat/non-flat regions or specially designed kernels to capture that behavior.

Furthermore, Bayesian optimization is inherently a local stationary process depending on the acquisition function. It has a dual behavior of global exploration and local exploitation. Ideally, both samples and uncertainty estimation end up being distributed unevenly, with many samples and small uncertainty near the local optima and sparse samples and large uncertainty everywhere else.

There has been several attempts to model nonstationary functions with Gaussian processes. For example, the use of specific nonstationary kernels [47], Bayesian treed GP models [22] or projecting (warping) the input space to a stationary latent space [49]. The idea of treed GPs was used in Bayesian optimization combined with an auxiliary local optimizer [59]. A version of the warping idea was applied to Bayesian optimization [54]. Later, Assael et al. [3] built a treed GPs where the warping model was used in the leaves. These methods try to model the nonstationary property in a global way. However, as pointed out before, sampling in Bayesian optimization is uneven, thus the global model might end up being inaccurate.

Like many global optimization and bandit setups, Bayesian optimization requires to control the bounds of the function to drive exploration efficiently. As pointed out in Bubeck et al. [8], tight bounds are required only near its optimum for these setups. Then, milder conditions can be defined elsewhere. For example, they use the “weak Lipschitz” condition as a milder version of the Lipschitz condition. In probabilistic terms, the upper and lower bounds defined by the Gaussian process in Bayesian optimization play the same role as the Lipschitz constant in classical optimization [14]. That is, high uncertainty represents loose bounds and viceversa. Our proposal exploits this idea by providing an adaptive kernel which allows tight bounds near the optimum and looser bounds everywhere else. As new information of the optimum location is available, the tight bounds are moved towards its location.

## 4 Spartan Bayesian Optimization

Our approach to nonstationarity is based on the model presented in Krause & Guestrin [35] where the input space is partitioned in different regions such as the resulting GP is the linear combination of local GPs:  $\xi(\mathbf{x}) = \sum_j \lambda_j(\mathbf{x})\xi_j(\mathbf{x})$ . Each local GP has its own specific hyperparameters, making the final GP nonstationary even when the local GPs are stationary. In order to achieve smooth interpolation between regions, the authors suggest the use of a weighting function  $\omega_j(\mathbf{x})$  for each region, having the maximum in region  $j$  and decreasing its value with distance to region  $j$  [35]. Then, we can set  $\lambda_j(\mathbf{x}) = \sqrt{\omega_j(\mathbf{x}) / \sum_p \omega_p(\mathbf{x})}$ . In practice, the mixed GP can be obtained by a combined kernel function of the form:  $k(\mathbf{x}, \mathbf{x}' | \boldsymbol{\theta}) = \sum_j \lambda_j(\mathbf{x})\lambda_j(\mathbf{x}')k_j(\mathbf{x}, \mathbf{x}' | \boldsymbol{\theta})$ . A related approach of additive GPs was used by Kandasamy et al. [32] for Bayesian optimization of high dimensional functions under the assumption that the actual function is a combination of lower dimensional functions.

For Bayesian optimization, we propose the combination of a local and a global kernels and with multivariate normal distributions as weighting functions. We have called this kernel, the Spartan kernel:

$$k(\mathbf{x}, \mathbf{x}' | \boldsymbol{\theta}^S) = \lambda^{(g)}(\mathbf{x})\lambda^{(g)}(\mathbf{x}')k^{(g)}(\mathbf{x}, \mathbf{x}' | \boldsymbol{\theta}^g) + \sum_{m=1}^M \lambda_m^{(l)}(\mathbf{x} | \boldsymbol{\theta}^p)\lambda_m^{(l)}(\mathbf{x}' | \boldsymbol{\theta}^p)k^{(l)}(\mathbf{x}, \mathbf{x}' | \boldsymbol{\theta}_m^l) \quad (11)$$

where the normalized local weights  $\lambda_m^{(l)}(\mathbf{x} | \boldsymbol{\theta}^p)$  includes parameters to move the influence region. The unnormalized weights  $\omega =$  are defined as:

$$\omega^{(g)} = \mathcal{N}(\psi, \mathbf{I}\sigma_g^2) \quad \omega_m^{(l)} = \mathcal{N}(\boldsymbol{\theta}^p, \mathbf{I}\sigma_m^2) \quad \forall m = 1 \dots M \quad (12)$$

where  $\psi$  and  $\boldsymbol{\theta}^p$  can be seen as the centers of the influence region of each kernel while  $\sigma_g^2$  and  $\sigma_m^2$  represents the area of influence. Note that all the local kernels share the same mean value (center), but different variance (area). Thus, it generates a funnel-like structure. The Spartan kernel with a single local kernel is shown in Figure 1.

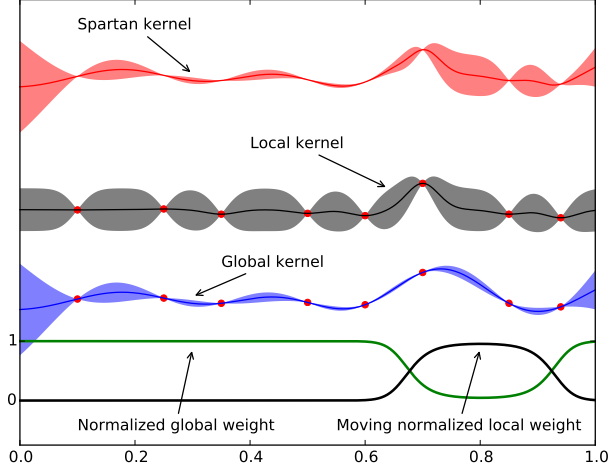


Figure 1: Representation of the Spartan kernel in SBO. In this case, the kernel is just a combination of a single local and a global kernel. Typically, the local and global kernels have a small and large length-scale respectively. The influence of each kernel is represented by the normalized weight at the bottom of the plot. Note how the kernel with small length-scale produce larger uncertainties which is an advantage for fast exploitation, but it can perform poorly for global exploration as it tends to sample equally everywhere. On the other hand, the kernel with large length-scale provides a better global estimate, but with is smoother with higher uncertainty which improves global exploration.

**Global kernel parameters** Unless we have prior knowledge of the function, the parameters of the global kernel are mostly irrelevant. In most applications, we can use a uniform distribution, which can be easily approximated with a large  $\sigma_g^2$ . For example, assuming a normalized input space  $\mathcal{X} = [0, 1]^d$ , we can set  $\psi = [0.5]^d$  and  $\sigma_g^2 = 10$ .

**Local kernel parameters** For the local kernels, we estimate the center of the funnel structure  $\theta^p$  based on the data gathered. We propose to consider  $\theta^p$  as part of the hyperparameters for the Spartan kernel, that also includes the parameters of the local and global kernels, that is,

$$\theta^S = [\theta^g, \theta_1^l, \dots, \theta_M^l, \theta^p]$$

The area of influence of each local kernel could also be adapted including the terms  $\sigma_m^2 \quad \forall m = 1 \dots M$  as part of the kernel hyperparameters  $\theta^S$ . However, in that case the problem becomes ill-posed, resulting in overfitting. Instead of adding regularization terms, we found simpler to fix the value of  $\sigma_m^2$  or define it in terms of numbers of samples inside. The second method has the advantage that, while doing exploitation, as the number of local samples increase, the funnel gets narrower, allowing better local refinement.

As commented in Section 2, when new data is available, all the parameters are updated using MCMC. Therefore, the position of the local kernel  $\theta^p$  is moved each iteration to represent the posterior. Due to the sampling behavior in Bayesian optimization, we found that it intrinsically moves more likely towards the more densely sampled areas in many problems, which corresponds to the location of the function minima. Furthermore, as we have  $m$  MCMC samples, there are  $m$  different positions for the local kernel  $\Theta^p = \{\theta_i^p\}_{i=1}^m$ . Algorithm 2 modifies algorithm 1 for Spartan Bayesian Optimization.

As can be seen in Algorithms 2, Spartan Bayesian Optimization is simple to implement once we have an implementation of Algorithm 1. It is important to note that, although we have implemented SBO relying on Gaussian processes and *expected improvement*, the funnel kernel also works with other popular models such as Student-t processes [65, 46], treed Gaussian processes [22] and other criteria such as *upper confidence bound* [55], *relative entropy* [26, 25], etc. In summary, it requires to modify the kernel function and hyperparameter estimation. However, as we will see in Section 5, the results show a large gain in terms of convergence speed and sample efficiency for many problems. The intuition behind SBO is the same of many acquisition functions in Bayesian optimization: *the aim of the surrogate model is not to approximate the target function precisely in every point, but to provide information about the*

---

**Algorithm 2** Spartan Bayesian Optimization (SBO)

---

```
1: Initial design of  $p$  points using any prior information (e.g.: LHS to maximize coverage)
2:  $\mathbf{X} \leftarrow \mathbf{x}_{1:p}$   $\mathbf{y} \leftarrow y_{1:p}$ 
3: for  $n = p \dots N$  do ▷ Available budget of  $N$  queries
4:   for  $i = 1 \dots m$  do ▷ Iterate over MCMC samples
5:      $k(\mathbf{x}, \mathbf{x}' | \boldsymbol{\theta}_i) \leftarrow \lambda_l(\mathbf{x} | \boldsymbol{\theta}_i^{pos}) \lambda_l(\mathbf{x}' | \boldsymbol{\theta}_i^{pos}) k_l(\mathbf{x}, \mathbf{x}' | \boldsymbol{\theta}_i^l) + \lambda_g(\mathbf{x}) \lambda_g(\mathbf{x}') k_g(\mathbf{x}, \mathbf{x}' | \boldsymbol{\theta}_i^g)$ 
6:      $\mu_i(\mathbf{x}) \leftarrow \mathbf{k}(\mathbf{x}, \mathbf{X} | \boldsymbol{\theta}_i) \mathbf{K}(\mathbf{X}, \mathbf{X} | \boldsymbol{\theta}_i)^{-1} \mathbf{y}$  ▷ Predicted mean
7:      $\sigma_i^2(\mathbf{x}) \leftarrow k(\mathbf{x}, \mathbf{x} | \boldsymbol{\theta}_i) - \mathbf{k}(\mathbf{x}, \mathbf{X} | \boldsymbol{\theta}_i) \mathbf{K}(\mathbf{X}, \mathbf{X} | \boldsymbol{\theta}_i)^{-1} \mathbf{k}(\mathbf{X}, \mathbf{x} | \boldsymbol{\theta}_i)$  ▷ Predicted variance
8:   end for
9:    $\boldsymbol{\Theta} \leftarrow \{\boldsymbol{\theta}_i^g, \boldsymbol{\theta}_i^l, \boldsymbol{\theta}_i^{pos}\}_{i=1}^m$  ▷ Update the hyperparameters using MCMC
10:   $\mathbf{x}_n = \arg \max_{\mathbf{x}} EI(\mathbf{x})$  ▷ Expected improvement, see (5)
11:   $y_n \leftarrow f(\mathbf{x}_n)$   $\mathbf{X} \leftarrow add(\mathbf{x}_n)$   $\mathbf{y} \leftarrow add(y_n)$ 
12: end for
```

---

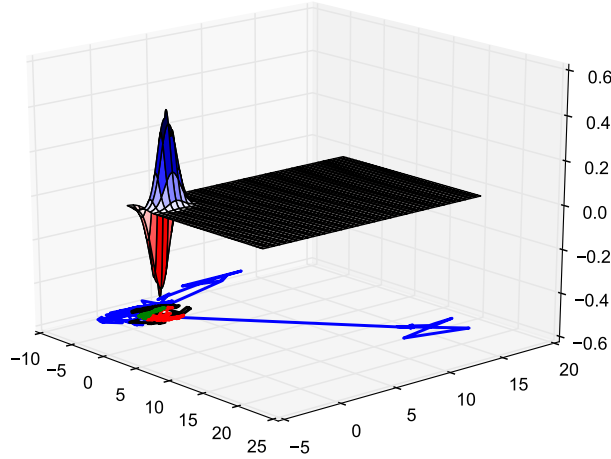


Figure 2: Left: Exponential 2D function from Gramacy [22]. The path below the surface represents the location of the local kernel as being sampled by MCMC. Clearly, it converges to the nonstationary section of the function. For visualization, the path is broken in colors by the order in the path (blue  $\rightarrow$  black  $\rightarrow$  green  $\rightarrow$  red).

*location of the minimum.* Many optimization problems are difficult due to the fact that the region near the minimum is heteroscedastic, i.e.: it has higher variability than the rest of the space, like the function in Figure 2. In this case, SBO greatly improves the performance of the state of the art in Bayesian optimization.

## 5 Evaluation and results

We have selected a variety of benchmarks from different applications in robotics to test our method. The purpose is twofold. First, we highly the potential applicability of Bayesian optimization in many ways in robotics, from design, control, software tuning, etc. Second, we show that in all those setups, our method outperforms the state of the art in Bayesian optimization.

1. We have taken several well-known functions for testing global optimization algorithms. The set of functions includes both stationary and non-stationary problems.
2. For the algorithm tuning and perception problems, we have selected a set of machine learning tuning benchmarks. They include both image classification and large clustering problems with different algorithms.
3. For the control/reinforcement learning problems, we have selected some classic benchmarks and a highly complex benchmark to control a hovering aerobatic helicopter.



4. Finally, we address the intelligent design and embodiment problem with the optimal design of a wing profile for a UAV. This is a highly complex scenario, due to the chaotic nature of fluid dynamics. Thus, this problem is ubiquitous in global optimization and evolutionary computation literature.

## 5.1 Implementation details

For evaluation purposes and to highlight the robustness of SBO, we have simplified the funnel structure to a single local and global kernel as in Figure 1. We also took the simpler approach to fix the variance of the  $\omega_l$ . We found that a single value of  $\sigma_l^2 = 0.05$  was robust enough in all the experiments once the input space was normalized to the unit hypercube.

Although this method allows for any combination of local and global kernels, for the purpose of evaluation, we used the Matérn kernel from equation (10) with automatic relevance determination for both –local and global– kernels. Furthermore, the kernel hyperparameters were initialized with the same prior for the both kernels. Therefore, we let the data determine which kernel has smaller length-scale. We found that the typical result is the behavior from Figure 1. However, in some problems, the method may learn a model where the local kernel has a larger length-scale (i.e.: smoother and smaller variance) than the global kernel, which may also improve the convergence in plateau-like functions. Besides, if the target function is stationary, the system might end up learning a similar length-scale for both kernels, thus being equivalent to a single kernel. We can say that SBO generalizes the modeling capabilities of standard BO, that is, standard BO is a special case of SBO where the local and global kernels are the same.

Given that for a single Matérn kernel with ARD, the number of kernel hyperparameters is the dimensionality of the problem,  $d$ , the number of hyperparameters for the Spartan kernel in this setup is  $3d$ . As we will see in the experiments, this is the only drawback of SBO compared to standard BO, as it requires running MCMC in a larger dimensional space, which results in higher computational cost. However, because SBO is more efficient, the extra computational cost can be easily compensated by a reduced number of samples.

We implemented<sup>2</sup> Spartan Bayesian Optimization (SBO) using the BayesOpt library [40] as codebase. This allowed us to evaluate the setup for many surrogates and criteria. For comparison, we also evaluated other nonstationary Bayesian optimization models:

- Input warping (WARP) from Snoek et al. [54] is the closest to our method both in terms of properties, applicability and results. To our knowledge, this is the only Bayesian optimization algorithm that has deal with nonstationarity using Gaussian processes in a fully correlated way. However, as presented in Section 5.6, WARP is much more expensive than SBO in terms of computational cost (at least, one order of magnitude in CPU time).
- Random forests [29] are excellent for discrete or categorical spaces and their computational (CPU time) performance is excellent. However, for continuous functions, their sample performance is not as good as any GP-based BO, as it has been reported previously in the literature [40, 15]. In Appendix B we discuss how to combine those two techniques to exploit the best properties of each method.
- Treed GPs [59, 3] can be quite expressive in terms of modeling. However, due to the reduced number of samples per leaf and reduced global correlation, we found that they require a larger nugget or noise term  $\sigma_n^2$  to avoid numerical instability. For deterministic functions or large signal to noise ratio, like those presented in the results, increasing the nugget reduces the accuracy of the results. Furthermore, as presented in Assael et al. [3], single nonstationary GPs like the input warping method or our Spartan kernel can be used as tree leaves. Thus, SBO can be considered as an enhancement of treed GPs, not a competitor.

For clarity in the plots, we have only included the results from WARP algorithm, as the current state of the art, and *standard* (stationary) BO, as a reference baseline. We did not include the random forests or treed GPs as its performance was worse than standard BO or WARP algorithms.

---

<sup>2</sup>The code will be release as open source when the paper gets published.

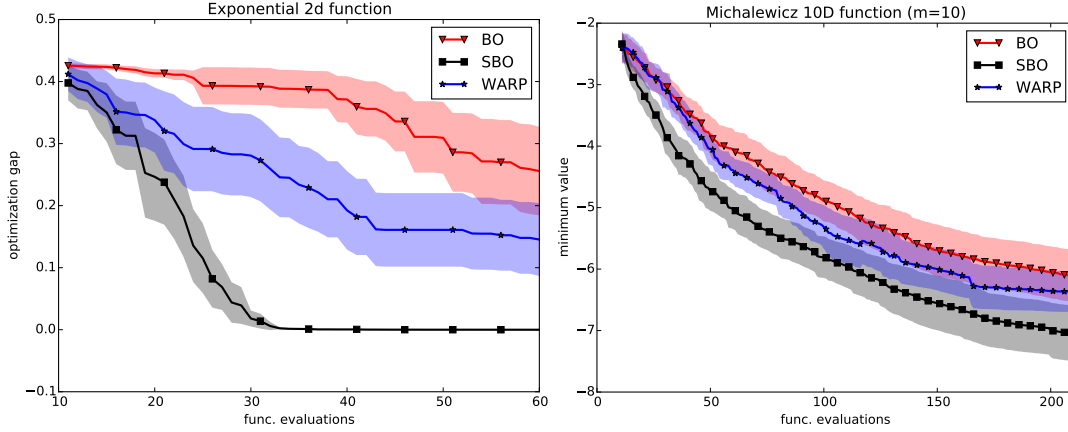


Figure 3: Left: Exponential 2D function from [22]. The proposed method *SBO* results in an outstanding convergence speed compared to the state of the art. *SBO* finds the minimum in about 30 function evaluations in all tests. Right: Michalewicz 10D function with  $m=10$ . The convergence is slow, but nonstationary methods clearly perform better with *SBO* being fastest.

For the experiments reported here we used: a Gaussian process with unit mean function like in [31]. For BO and WARP we also used a Matérn kernel  $\nu = 5/2$  with ARD. The kernel hyperparameters, including  $\theta^p$  for SBO and  $(\alpha, \beta)$  for the warping functions were estimated using MCMC (i.e.: slice sampling). Due to the computational burden of MCMC, we used a small number of samples (i.e.: 10), while trying to decorrelate every resample with large burn-in periods (i.e.: 100 samples) as in Snoek et al. [52]. All experiments were repeated 20 times using common random numbers. As commented in Section 2, the number of function evaluations in each plot includes an initial design of 10 points using *latin hypercube sampling*.

## 5.2 Optimization benchmarks

We evaluated the algorithms on a set of well-known test functions for global optimization both smooth or with sharp drops.

Figure 3 shows the results for the optimization benchmarks for functions with sharp drops where local optimization is fundamental. First, the exponential 2D function from Gramacy [22]:

$$f(\mathbf{x}) = x_1 \exp(-x_1^2 - x_2^2) \quad (13)$$

with  $x_1, x_2 \in [-2, 18]^2$ . Our method (*SBO*) provides excellent results, by reaching the optimum in less than 25 iterations (35 samples) for all tests. Because the function is nonstationary, the WARP method outperforms standard BO, but its convergence is much slower,

Figure 3, also shows the results for the Michalewicz function. The function has a parameter to define the dimensionality  $d$  and the steepness  $m$ . The function is defined as:

$$f(\mathbf{x}) = \sum_{i=1}^d \sin(x_i) \sin^{2m} \left( \frac{ix_i^2}{\pi} \right) \quad (14)$$

for  $\mathbf{x} \in [0, \pi]^d$ . This function is known to be a hard benchmarks in global optimization due to the many local minima ( $d!$ ) and steep drops. We used  $d = 10$  and  $m = 10$ , resulting in 3628800 minima with very steep edges. For this problem, *SBO* clearly outperforms the rest of the methods by a large margin.

Figure 4 show the results functions with smooth or wide minima. Bayesian optimization is more suitable for these function and naive strategies perform well in general. Therefore, there is barely room from improvement. However, we show that, even in this situation, *SBO* is equal or better than stationary BO. There is no penalty for the extra complexity on the model. However, the WARP method may get slower convergence due to the extra complexity. First, we evaluated the Branin-Hoo function, which has

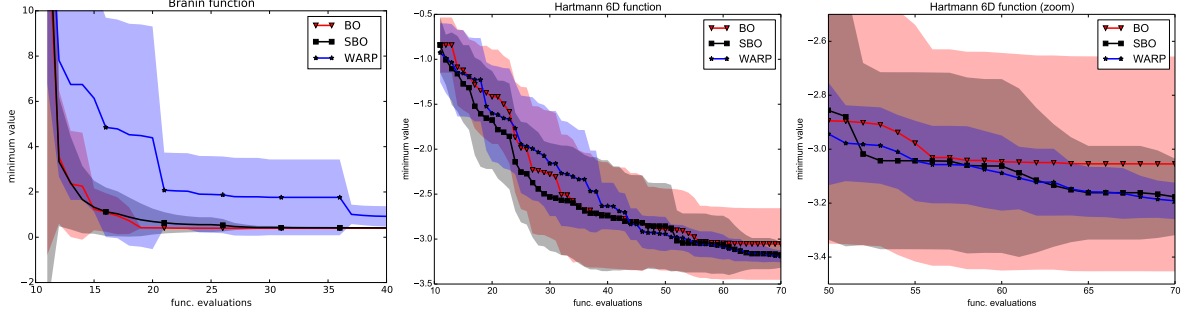


Figure 4: Left: Minimum value for the Branin-Hoo function. BO and SBO are barely identical. Center: Hartmann 6D function. Evolution of the minimum value for each method. Right: Zoom of the minimum value during the final iterations. We can see how both SBO and WARP produce more accurate results and with smaller uncertainty, meaning that the results are more repeatable.

become a standard benchmark in Bayesian optimization,

$$f(\mathbf{x}) = \left( x_2 - \frac{5.1}{4\pi^2} x_1^2 + \frac{5}{\pi} x_1 - 6 \right)^2 + 10 \left( 1 - \frac{1}{8\pi} \right) \cos(x_1) + 10 \quad (15)$$

with  $x_1 \in [-5, 10]$  and  $\mathbf{x}_2 \in [0, 15]$ . As can be seen in Figure 4, the differences between SBO and BO are insignificant. Meanwhile, the warping function in WARP introduces an exploration bias at early stages, resulting in slower convergence.

Finally, we tested the Hartmann 6D function for  $\mathbf{x} \in [0, 1]^6$ :

$$f(\mathbf{x}) = - \sum_{i=1}^4 \alpha_i \exp \left( - \sum_{j=1}^6 A_{ij} (x_j - P_{ij})^2 \right) \quad (16)$$

with  $\alpha$ ,  $A$  and  $P$  defined in Appendix A. In this case, the differences are small, which imply that the function is most likely stationary and simple to exploit. However, we can see in the variance of the plots that nonstationary methods are more robust during the final exploitation.

### 5.3 Benchmarks for machine learning hyperparameter tuning

Our next set of experiments was based on a set of problems for automatic tuning of machine learning algorithms.

The first problem consists on tuning the 4 parameters of a *logistic regression* classifier to recognize handwritten numbers from the MNIST dataset (see Figure 6). As can be seen in the results, it is an easy problem for Bayesian optimization. Even the standard BO method were able to reach the minimum in less than 50 function evaluations. In this case, the warped method was the fastest one, with almost 20 evaluations. The proposed method had similar performance in terms of convergence, but with one order of magnitude lower execution time (see Section 5.6).

The second problem is called *online LDA*. This problem is based on the setup defined in Snoek et al [52] for learning topics of Wikipedia articles using Latent Dirichlet Allocation in an online fashion. It requires to tune 3 parameters. Both the standard BO and the WARP method got stuck while our method was able escape from the local minimum and outperform other methods by a large margin. Again, SBO required much lower computational cost than WARP.

Finally, we evaluated the *HP-NNET* problem, based on a deep neural network written by Bergstra et al. [4] to classify a modified MNIST dataset. In this case, the handwritten numbers were arbitrarily rotated and with random background images as distractors (see Figure 6). The new database is called MRBI, for MNIST Rotated and with Background Images. In this case, due to the high dimensionality and heterogeneity of the input space (7 continuous + 7 categorical parameters) we tested two approaches.

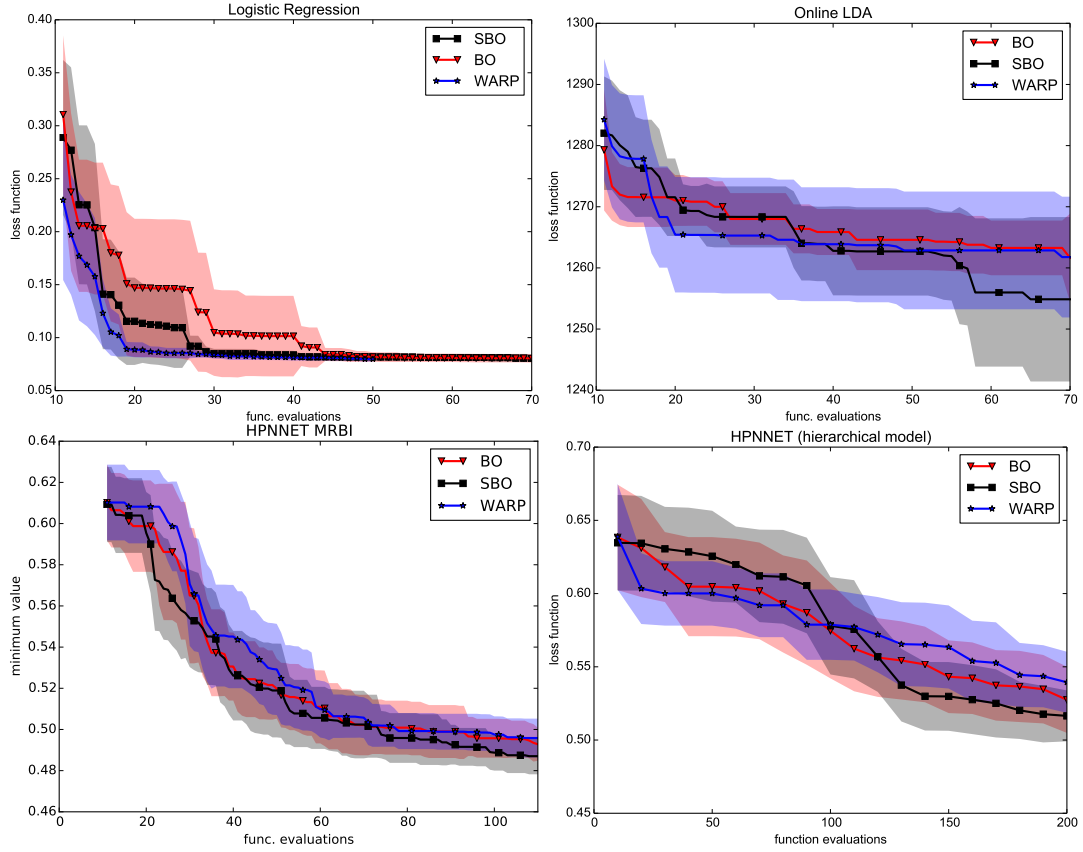


Figure 5: Surrogate benchmarks by [15] based on real hyperparameter optimization of machine learning algorithms. From left to right: logistic regression (4D continuous), online LDA (3D continuous), deep neural network (HP-NNET with the mrbi dataset, 7D continuous, 7D categorical), and deep neural network using the hierarchical model of Appendix B. In all cases SBO performs better than BO. Only in the logistic regression problem, the WARP method outperforms SBO.



Figure 6: The classifications problems use images of handwritten numbers. The objective of the classifier is to identify the correct number. Left: MNIST dataset of handwritten numbers used for the logistic regression problem. Right: Modified MNIST dataset with Rotated numbers and Background Images (MRBI) to increase the difficulty of classification. It was used in the HP-NNET problem.

First, we applied a *single fully-correlated model* for all the variables. The categorical variables were mapped to integer values that were computed by rounding the query values. In this case, similarly to the Hartmann function, our method (SBO) is more precise and robust.

Second, we applied a *hierarchical model* (see Appendix B for the details), which splits the continuous and categorical variables in a two layer optimization process. In this case, the nonstationary algorithms (SBO or WARP) were only applied on the continuous variables (outer loop). For the inner loop we used a Hamming kernel for categorical variables [28, 62].

$$k_H(\mathbf{x}, \mathbf{x}' | \theta) = \exp \left( -\frac{\theta}{2} g(s(\mathbf{x}), s(\mathbf{x}'))^2 \right) \quad (17)$$

where  $s(\cdot)$  is a function that maps continuous vectors to discrete vectors by scaling and rounding. The function  $g(\cdot, \cdot)$  is defined as the Hamming distance  $g(\mathbf{x}, \mathbf{x}') = |\{i : x_i \neq x'_i\}|$  so as not to impose an artificial ordering between the values of categorical parameters.

Note that the plots are with respect to target function evaluations. However, the results of the hierarchical model are based on only 20 iterations of the outer loop, as each iteration requires 10 function evaluations in the inner loop. At early stages, SBO was trying to find a good location for the local kernel and the performance was slightly worse. However, after some data was gathered, the local kernel jumped to a good spot and the convergence was faster.

For these benchmarks, due to the complexity of the machine learning algorithms and the size of the datasets a single evaluation can take hours or days of wall-time. In order to simplify the comparison and run more tests, we used the surrogate benchmarks presented in Eggenberger et al. [15]. It uses surrogate functions build with real data and evaluations to predict the performance of the tuned algorithms. Evaluating the surrogate function can be done in seconds, compared to the actual training each machine learning algorithm. The authors provide different surrogates [15]. We selected the Gradient Boosting as it provides the lowest prediction error (RMSE) with respect to the actual data from each problem. We explicitly rejected the Gaussian Process surrogate benchmark to avoid the potential advantage of perfectly modeling. The results are shown in Figure 5. A detailed description of the experiments can be found in the original paper [15].

## 5.4 Reinforcement learning benchmarks

We evaluated SBO with several reinforcement learning problems. Reinforcement learning deals with the problem of how artificial agents perform optimal behaviors. An agent is defined by a set of variables  $\mathbf{s}_t$  that capture the current state and configuration of the agent in the world. The agent can then perform an action  $\mathbf{a}_t$  that modifies the agent state  $\mathbf{s}_{t+1} = T(\mathbf{s}_t, \mathbf{a}_t)$ . At each time step, the agent collects the reward signal associated with its current state and action  $R_t$ . The actions are selected according to a policy function that represents the agent behavior  $\mathbf{a}_{t+1} = \pi(\mathbf{s}_t)$ . The states, actions and transitions are modeled using probabilities to deal with uncertainty. Thus, the objective of reinforcement learning is to find the optimal policy  $\pi^*$  that maximizes the expected reward for a defined time horizon

$$\pi^* = \arg \max_{\pi} \mathbb{E}_{\mathbf{s}_{0:T}, \mathbf{a}_{1:T}} \left[ \sum_{t=1}^T r_t \right] \quad (18)$$

In the problems we have selected, we assume a finite time horizon. The expectation is evaluated using Monte Carlo, by sampling several episodes for a given policy.

Reinforcement learning algorithms usually rely on variants of the Bellman equation to optimize the policy step by step considering each instantaneous reward  $r_t$  separately. Some algorithms also rely on partial or total knowledge of the transition model  $\mathbf{s}_{t+1} = T(\mathbf{s}_t, \mathbf{a}_t)$  in advance. Other methods tackle the optimization problem directly, considering equation (18) as an stochastic optimization problem. Those methods are called *direct policy search* [66]. The use of Bayesian optimization for reinforcement learning was previously called *active policy search* [42] for its connection with active learning and how samples are carefully selected based on current information.

The main advantage of using Bayesian optimization to compute the optimal policy is that it can be done with very little information. In fact, as soon as we are able to simulate scenarios and return the total reward  $\sum_{t=1}^T r_t$ , we do not need to access the dynamics, the instantaneous reward or the current

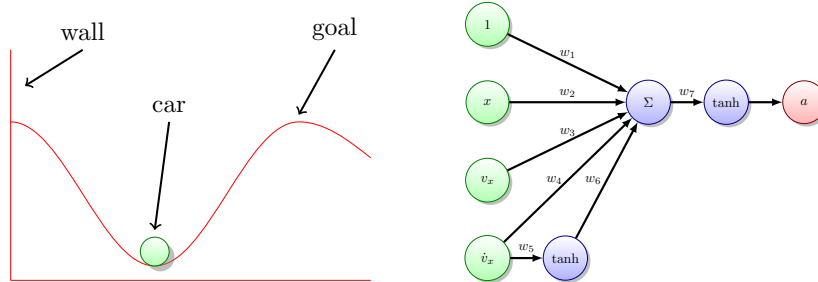


Figure 7: Left: Mountain car scenario. The car is underactuated and cannot climb to the goal directly. Instead it requires to move backwards to get inertia. The line in the left is an inelastic wall. Right: Policy use to control the mountain car. The inputs are the horizontal position  $x_t$ , velocity  $v_t = x_t - x_{t-1}$  and acceleration  $a_t = v_t - v_{t-1}$  of the car. The output is the car throttle bounded to  $[-1, 1]$ .

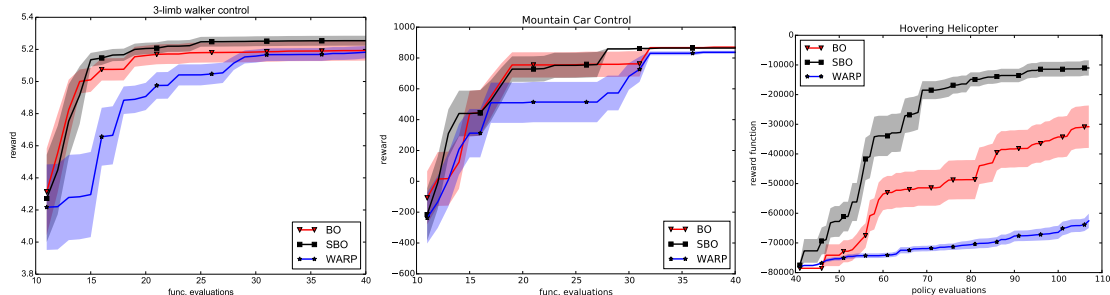


Figure 8: Total reward for the three limb walker (left), the mountain car and the hovering helicopter problem. For the first problem, SBO is able to achieve higher reward, while other methods get stuck in a local maxima. For the mountain car, SBO is able to achieve maximum performance in all trials after just 27 policy trials (17 iterations + 10 initial samples). For the helicopter problem, BO and WARP have slow convergence, because many policies results in an early crash. Providing almost no information. However, SBO is able to exploit good policies and quickly improve the performance.

state of the system. Furthermore, there is no need for space or action discretization, building complex features or *tile coding* [57]. We found that for many problems, a simple, low dimensional, quase-linear controller is able to achieve state-of-the-art performance if properly optimized.

A frequent issue for applying general purpose optimization algorithms for policy search is that, in many problems, the occurrence of *failure* states or scenarios results in large discontinuities or flat regions due to large penalties. This is opposed to the behavior of the reward near the optimal policy where small variations on a suboptimal policy can considerably change the performance achieved. Therefore, the resulting reward function presents a nonstationary behavior with respect to the policy.

We have compared our method in three well-known benchmarks with different level of complexity. The first problem is learning the controller of a three limb robot walker presented in Westervelt et al. [64]. The controller modulates the walking pattern of a simple biped robot. The desired behavior is a fast upright walking pattern, the reward is based on the walking speed with a penalty for not maintaining the upright position. The dynamic controller has 8 continuous parameters. The walker problem was already used as a Bayesian optimization benchmark [26].

The second problem is the mountain car problem represented in Figure 7. The state of the system is the car horizontal position. The action is the horizontal acceleration  $a \in [-1, 1]$ . Contrary to the many solutions that discretize both the state and action space, we can directly deal with continuous states and actions. The policy is a perceptron model inspired by Brochu et al. [6] as can be seen in Figure 7. The potentially unbounded policy parameters  $\mathbf{w} = \{w_i\}_{i=1}^7$  are computed as

$$\mathbf{w} = \tan \left( (\pi - \epsilon_\pi) \mathbf{w}_{01} - \frac{\pi}{2} \right)$$

where  $\mathbf{w}_{01}$  are the policy parameters bounded in the  $[0, 1]^7$  space. The term  $\epsilon_\pi$  was used to avoid  $w_i \rightarrow \infty$ . For the experiments, we used  $\epsilon_\pi = 10^{-4}$ .

The third problem is the hovering helicopter from the RL-competition<sup>3</sup>. This is one of the most challenging scenarios of the competition, being presented in all the editions. This problem is based on a simulator of the XCell Tempest aerobatic helicopter. The simulator model was learned based on actual data from the helicopter using apprenticeship learning [1]. The model was used to learn a policy that was later used in the real robot. The simulator included several difficult wind conditions. The state space is 12D (position, orientation, translational velocity and rotational velocity) and the action is 4D (forward-backward cyclic pitch, lateral cyclic pitch, main collective pitch and tail collective pitch). The reward is a quadratic function that penalizes both the state error (inaccuracy) and the action (energy). Each episode is run during 10 seconds (6000 control steps). If the simulator enters a terminal state (crash), a large negative reward is given, corresponding to getting the most negative reward achievable for the remaining time.

We also used a weak baseline controller that was included with the helicopter model. This weak controller is a simple linear policy with 12 parameters (weights). In theory, this controller is enough to avoid crashing but is not very robust. We show how this policy can be easily improved with few iterations. In this case, initial exploration of the parameter space is specially important because the number of policies not crashing in few control steps is very small. For most policies, the reward is the most negative reward achievable. Thus, in this case, we have used Sobol sequences for the initial samples of Bayesian optimization. These samples are deterministic, therefore we guarantee that the same number of non-crashing policies are sampled for every trial and every algorithm. We also increased the number of initial points to 40.

Figure 8 shows the performance for the three limb walker presented, the mountain car and the helicopter problem. In all cases, the results obtained by SBO were more efficient in terms on number of trials and accuracy, with respect to standard BO and WARP. Furthermore, we found that the results of SBO were comparable to those obtained by popular reinforcement learning solvers like SARSA [57], but with much less information and prior knowledge about the problem. For the helicopter problem, other solutions found in the literature require a larger number of scenarios/trials to achieve similar performance [2, 34].

## 5.5 Automatic wing design using a CDF software

Computational fluid dynamics (CDF) software is a powerful tool for the design of mechanical and structural elements subject to interaction with fluids, such as aerodynamics, hydrodynamics or heating systems. Compared to physical design and testing in wind tunnels or artificial channels, the cost of simulation is almost negligible. Because simulated redesign is simple, CDF methods have been used for autonomous design of mechanical elements following principles from experimental design. This simulation-based autonomous design is a special case of the design and analysis of computer experiments (DACE). Recent developments both in terms of algorithms and available software have reduced the computational cost and improved the accuracy of the results, which allows DACE methodologies to be used to find optimal designs based on trial and error. Nevertheless, the computational cost of an average CDF simulation can still take days or months of CPU time. Therefore, the use of a sample efficient methodology is mandatory. Bayesian optimization has an enormous potential in the field for autonomous design.

The experiment that we designed to highlight the potential of Bayesian optimization in this field was the autonomous design of an optimal wing for an airplane [17]. We simulated a wind tunnel using the xFlow<sup>TM</sup>CDF software. The objective of the experiment was to find the shape of the wing that minimizes the drag while maintaining enough lift. First, as a common practice in this kind of problems, we assumed a 2D simulation of the fluid along the profile of the wing. This is a reasonable assumption for wings with large aspect ration (large span compared to the chord), and it considerably reduces the wall time of each simulation from days to hours. For the parametrization of the profile, there are many alternatives based on geometric or manufacturing principles. In our case, we used Bezier curves for its simplicity to generate the corresponding shape. However, note that Bayesian optimization is agnostic of the geometric parametrization and any other parametrization could also be used. The Bezier curve of the wing was based on 7 control points, which resulted in 14 parameters. However, adding some physical and manufacturing restrictions resulted in 5 free parameters.

---

<sup>3</sup><http://www.rl-competition.org/>

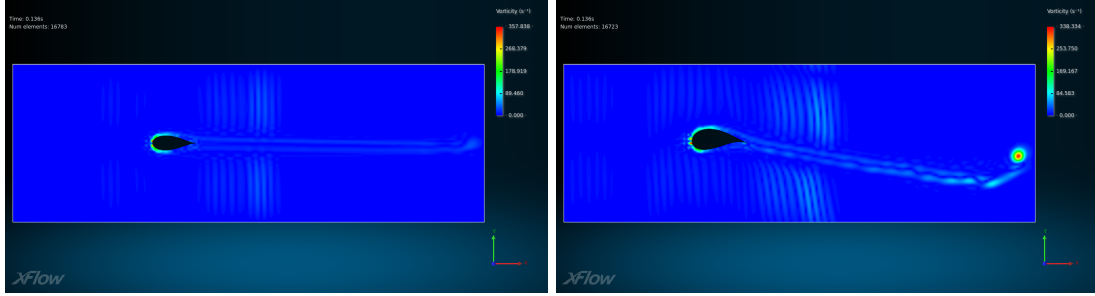


Figure 9: Vorticity plots of two different wing shapes. The left wing barely affect the trajectory of the wind, resulting in not enough lift. Meanwhile the right wing is able to provide enough lift with minimum drag.

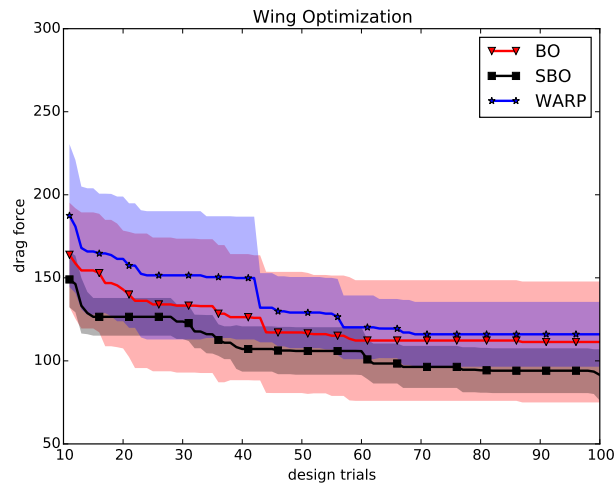


Figure 10: Results for the wing design optimization. Each plot is based on 10 runs.



Table 1: Average total CPU time in seconds.

Time (s)	Exp2	Branin	Hart.	Michal.	Walker	MCar	LogR	OnLDA	HPNN	HPNN-h
#dims	2	2	6	10	8	7	4	3	14	7+7
#evals	60	40	70	210	40	40	50	70	110	200 (20)
BO	120	171	460	8 360	47	38	28	112	763	20
SBO	2 481	3 732	10 415	225 313	440	797	730	2 131	28 442	146
WARP	13 929	28 474	188 942	4 445 854	20 271	18 972	9 149	21 299	710 974	2 853

The problem of minimizing the drag directly is that the best solutions tends to generate flat wings that do not provide enough lift for the plane. Figure 9 shows comparison of a wing with no lift and the optimal design. Although there has been some recent work on Bayesian optimization with constraints [19, 21] we decided to use a simpler approach of adding a penalty with two purposes. First, the input space remains unconstrained, improving the performance of the optimization of the acquisition function. Second, due to the kernel smoothing, each penalty points generates a *safety area* where the function is partly penalized. Besides the drops and edges due to penalties, fluid dynamics also have chaotic properties: small changes in the initial conditions may produce a large variability in the result. For example, the flow near the trailing edge can transition from laminar to turbulent regime due to a small change in the wing shape. Thus, the resulting forces are completely different, increasing the drag and reducing the lift. Under this conditions, Figure 10 shows how both BO and WARP fail to find the optimum wing shape. However, SBO finds a better wing shape. Furthermore, it does it in few iterations.

## 5.6 Computational cost

Table 1 shows the average CPU time of the different experiments for the total number of function evaluations. HPNN-h uses the hierarchical model, with 200 function evaluations but only 20 iterations of the optimization loop, thus being faster. Due to the extensive evaluation, we had to rely on different machines for running the experiments, although all the algorithms for a single experiment were compared on the same machine. Thus, CPU time of different experiments might not be directly comparable.

The main difference between the three methods in terms of the algorithm is within the kernel function  $k(\cdot, \cdot)$ , which includes the evaluation of the weights in SBO and the evaluation of the warping function (the cumulative density function of a Beta distribution or Beta CDF) in WARP. The rest of the algorithm is equivalent. That is, we used the same code optimization techniques. For that reason, we decided to measure the CPU time as a direct metric of the computational cost. Besides, CPU time is more robust than wall-time.

After some profiling, we found that the time differences between the algorithms were mainly driven by the dimensionality of the hyperparameter space because MCMC was the main bottleneck. Furthermore, the shape of the posterior distribution of the kernel hyperparameters (which is sampled through MCMC) played an important role. The likelihood of the parameters for the Beta CDF was very narrow and the slice sampling algorithm spent many iterations before accepting a samples. Narrow likelihoods and large-flip sampling are well known issues for MCMC. There are methods that could be applied to alleviate the issues such as hybrid Monte Carlo or sequential Monte Carlo samplers, but that remains an open problem. Note that, for all algorithms and experiments, we used slice sampling as recommended by the authors of WARP [54]. Finally, the evaluation of the Beta CDF was much more involved and computationally expensive than the evaluation of the Matérn kernel or the Gaussian weights. That extra cost became an important factor as the kernel function was being called billions of times for each Bayesian optimization run.

It is important to note that, although Bayesian optimization is intended for expensive functions and the cost per iteration is negligible in most applications (for example: CDF simulations, deep learning algorithm tuning, etc.), the difference between methods could mean hours of CPU-time for a single iteration, changing the range of potential applications and inexpensive function evaluations.

## 6 Conclusions

In this paper, we have presented a new algorithm called Spartan Bayesian Optimization (SBO) which combines a local and a global kernel in a single adaptive kernel to deal with the exploration/exploitation

trade-off and the inherent nonstationarity in the search process during Bayesian optimization. We have shown that this new kernel increases the convergence speed and reduces the number of samples in many kind of problems. For nonstationary problems, the method provides excellent results compared to standard Bayesian optimization and the state of the art method to deal with nonstationarity. Furthermore, SBO also performs well in stationary problems by improving local refinement while retaining global exploration capabilities. We evaluated the algorithm extensively in standard optimization benchmarks, automatic wing design and machine learning applications, such as hyperparameter tuning problems and classic reinforcement learning scenarios. The results have shown that SBO outperforms the state of the art in Bayesian optimization for all the experiments and tests. It requires less samples or achieves smaller optimization gap. In addition to that, we have shown how SBO was much more efficient in terms of CPU usage than other nonstationary methods for Bayesian optimization.

The results in reinforcement learning also highlight the potential of the *active policy search* paradigm for reinforcement learning. Our method is specially suitable for that paradigm. This fact opens a new opportunity for Bayesian optimization as an efficient and competitive reinforcement learning solver, without relying on the dynamics of the system, instantaneous rewards or discretization of the different spaces.

## A Parameters of the Hartmann 6D function

The Hartmann 6D function for  $\mathbf{x} \in [0, 1]^6$  is defined as:

$$f(\mathbf{x}) = - \sum_{i=1}^4 \alpha_i \exp \left( - \sum_{j=1}^6 A_{ij} (x_j - P_{ij})^2 \right) \quad (19)$$

with

$$\begin{aligned} \alpha &= (1.0, 1.2, 3.0, 3.2)^T, \\ A &= \begin{pmatrix} 10 & 3 & 17 & 3.5 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{pmatrix}, \\ P &= 10^{-4} \begin{pmatrix} 1312 & 1696 & 5569 & 124 & 8283 & 5886 \\ 2329 & 4135 & 8307 & 3736 & 1004 & 9991 \\ 2348 & 1451 & 3522 & 2883 & 3047 & 6650 \\ 4047 & 8828 & 8732 & 5743 & 1091 & 381 \end{pmatrix} \end{aligned}$$

## B Discussion on mixed input spaces in Bayesian optimization

Although Bayesian optimization started as a method to solve classic nonlinear optimization problems with box-bounded restrictions, its sample efficiency and the flexibility of the surrogate models have attracted the interest of other communities and expanded the potential applications of the method.

In many current Bayesian optimization applications, like hyperparameter optimization, it is necessary to simultaneously optimize different kinds of input variables, for example: continuous, discrete, categorical, etc. While Gaussian processes are suitable for modeling those spaces by choosing a suitable kernel [58, 28], Bayesian optimization can become quite involved as the acquisition function (criterion) must be optimized in the same mixed input space. Available implementations of Bayesian optimization like Spearmint [52] use grid sampling and rounding tricks to combine different input spaces. However, this reduces the quality of the final result compared to proper nonlinear optimization methods [40]. Other authors have proposed some heuristics specially designed for criterion maximization in Bayesian optimization [14], but its applicability to mixed input spaces still remains an open question.

We propose a *hierarchical Bayesian optimization model*, where the input space  $\mathcal{X}$  is partitioned between homogeneous variables, for example: continuous variables  $\mathbf{x}^c$  and discrete variables  $\mathbf{x}^d$ . That is:

$$\mathcal{X} = \mathcal{X}^c \cup \mathcal{X}^d \doteq \{\mathbf{x} = [\mathbf{x}^c, \mathbf{x}^d]^T : \mathbf{x}^c \in \mathcal{X}^c \vee \mathbf{x}^d \in \mathcal{X}^d\} \quad (20)$$

Therefore, the evaluation of an element higher in the hierarchy implies the full optimization of the elements lower in the hierarchy. In principle, that would require many more function evaluations but, as the input space has been partitioned, the dimensionality of each separate problem is much lower. In practice, for the same number of function evaluations, the computational cost of the optimization algorithm is considerably reduced. We can also include conditional variables in the outer loop to select which computations to perform in the inner loop.

---

**Algorithm 3** Hierarchical Bayesian Optimization

---

```

1: Total budget  $N = N_c \cdot N_d$  ▷ Discrete iterations times continuous iterations
2: Split the discrete and continuous components of input space  $\mathbf{x} = [\mathbf{x}^c, \mathbf{x}^d]^T$ 
3: Initial samples for  $\mathbf{x}^c$ 
4: for  $n = 1 \dots N_c$  do ▷ Outer loop
5:   Update model (e.g.: Gaussian process) for  $\mathbf{x}^c$ 
6:   Find continuous component of next query point  $\mathbf{x}_n^c$  (e.g.: maximize EI)
7:   Initial samples for  $\mathbf{x}^d$ 
8:   for  $k = 1 \dots N_d$  do ▷ Inner loop
9:     Update model (e.g.: Random forest) for  $\mathbf{x}^d \mid \mathbf{x}_n^c$ 
10:    Find discrete component of next query point  $\mathbf{x}_k^d \mid \mathbf{x}_n^c$ 
11:    Combine queries and evaluate function:  $y_k \leftarrow f([\mathbf{x}_n^c, \mathbf{x}_k^d])$ 
12:  end for
13:  Return optimal discrete query for current continuous query:  $\mathbf{x}_n^{*(d)} \mid \mathbf{x}_n^c$ 
14: end for
15: Return optimal continuous query and corresponding discrete match:  $\mathbf{x}^* = [\mathbf{x}^{*(c)}, \mathbf{x}^{*(d)}]^T$ 

```

---

An advantage of this approach is that we can combine different surrogate models for different levels of the hierarchy. For example, using Random Forests [29] or tree-structured Parzen estimators [4] could be more suitable as a surrogate model for certain discrete/categorical variables than Gaussian processes. We could also use specific kernels like the Hamming kernel as we used in the previous section.

In contrast, we loose the correlation among variables in the inner loop, which may be counterproductive in certain situations. A similar alternative in the case where the target function is actually a combination of lower spaces, could be to use additive models, such as additive GPs [32].

Figure 5 shows how the method requires more function evaluations to achieve similar results than the fully correlated approach. However, as can be seen in Table 1, the computations cost and number of iterations increases, which might open new applications. A similar approach has been recently proposed to deal with high dimensional problems where evaluations are not expensive [61].

## References

- [1] Pieter Abbeel, Adam Coates, Morgan Quigley, and Andrew Ng. An application of reinforcement learning to aerobatic helicopter flight. In *Advances in Neural Information Processing Systems*, 2006.
- [2] Anwarissa Asbah, André MS Barreto, Clement Gehring, Joelle Pineau, and Doina Precup. Reinforcement learning competition: Helicopter hovering with controllability and kernel-based stochastic factorization. In *International Conference on Machine Learning (ICML), Reinforcement Learning Competition Workshop*, 2013.
- [3] John-Alexander M. Assael, Ziyu Wang, and Nando de Freitas. Heteroscedastic treed bayesian optimisation. Technical report, arXiv, 2014.
- [4] J. Bergstra, R. Bardenet, Y. Bengio, and B. Kgl. Algorithms for hyper-parameter optimization. In *NIPS*, pages 2546–2554, 2011.
- [5] Ali Borji and Laurent Itti. Bayesian optimization explains human active search. In C.J.C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 55–63. Curran Associates, Inc., 2013.

- [6] E. Brochu, V.M. Cora, and N. de Freitas. A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. eprint arXiv:1012.2599, arXiv.org, December 2010.
- [7] Eric Brochu, Tyson Brochu, and Nando de Freitas. A Bayesian interactive optimization approach to procedural animation design. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2010.
- [8] Sébastien Bubeck, Rémi Munos, Gilles Stoltz, and Csaba Szepesvári. X-armed bandits. *Journal of Machine Learning Research*, 12:1655–1695, 2011.
- [9] Adam D. Bull. Convergence rates of efficient global optimization algorithms. *Journal of Machine Learning Research*, 12:2879–2904, 2011.
- [10] R. Calandra, A. Seyfarth, J. Peters, and M. Deisenroth. Bayesian optimization for learning gaits under uncertainty. *Annals of Mathematics and Artificial Intelligence (AMAI)*, 1 1:1–19 1–19, 2015.
- [11] Antoine Cully, Jeff Clune, Danesh Tarapore, and Jean-Baptiste Mouret. Robots that can adapt like animals. *Nature*, 521:503507, 2015.
- [12] Wojciech M Czarnecki, Sabina Podlowska, and Andrzej J Bojarski. Robust optimization of svm hyperparameters in the classification of bioactive compounds. *Journal of cheminformatics*, 7(1):1–15, 2015.
- [13] C. Daniel, O. Kroemer, M. Viering, J. Metz, and J. Peters. Active reward learning with a novel acquisition function. *Autonomous Robots*, 39(3):389–405, 2015.
- [14] Nando de Freitas, Alex Smola, and Masrour Zoghi. Exponential regret bounds for Gaussian process bandits with deterministic observations. In *International Conference on Machine Learning (ICML)*, 2012.
- [15] K. Eggersperger, F. Hutter, H.H. Hoos, and K. Leyton-Brown. Efficient benchmarking of hyperparameter optimizers via surrogates. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence*, January 2015.
- [16] Matthias Feurer, Aaron Klein, Katharina Eggersperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems*, pages 2944–2952, 2015.
- [17] Alexander IJ Forrester, Neil W Bressloff, and Andy J Keane. Optimization using surrogate models and partially converged computational fluid dynamics simulations. *Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*, 462(2071):2177–2204, 2006.
- [18] Joerg M Gablonsky and C Tim Kelley. A locally-biased form of the DIRECT algorithm. *Journal of Global Optimization*, 21(1):27–37, 2001.
- [19] Jacob Gardner, Matt Kusner, Zhixiang Xu, Kilian Weinberger, and John Cunningham. Bayesian optimization with inequality constraints. In *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, pages 937–945, 2014.
- [20] Roman Garnett, Michael A. Osborne, and Stephen J. Roberts. Bayesian optimization for sensor set selection. In *Information Processing in Sensor Networks (IPSN 2010)*, 2010.
- [21] Michael A. Gelbart, Jasper Snoek, and Ryan P. Adams. Bayesian optimization with unknown constraints. In *Uncertainty in Artificial Intelligence (UAI)*, 2014.
- [22] Robert B Gramacy. *Bayesian treed Gaussian process models*. PhD thesis, University of California, Santa Clara, 2005.
- [23] Robert B. Gramacy and Herbert K.H. Lee. Cases for the nugget in modeling computer experiments. *Statistics and Computing*, 22:713–722, 2012.

- [24] H-M Gutmann. A radial basis function method for global optimization. *Journal of Global Optimization*, 19(3):201–227, 2001.
- [25] Philipp Hennig and Christian J. Schuler. Entropy search for information-efficient global optimization. *Journal of Machine Learning Research*, 13:1809–1837, 2012.
- [26] José Miguel Hernández-Lobato, Matthew W Hoffman, and Zoubin Ghahramani. Predictive entropy search for efficient global optimization of black-box functions. In Z. Ghahramani, M. Welling, C. Cortes, N.D. Lawrence, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 918–926. Curran Associates, Inc., 2014.
- [27] D. Huang, T. T. Allen, W. I. Notz, and N. Zheng. Global optimization of stochastic black-box systems via sequential kriging meta-models. *Journal of Global Optimization*, 34(3):441–466, 2006.
- [28] Frank Hutter. *Automated Configuration of Algorithms for Solving Hard Computational Problems*. PhD thesis, University of British Columbia, Vancouver, Canada, 2009.
- [29] Frank Hutter, Holger H. Hoos, and Kevin Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *LION-5*, page 507523, 2011.
- [30] Donald R. Jones. A taxonomy of global optimization methods based on response surfaces. *Journal of Global Optimization*, 21:345–383, 2001.
- [31] Donald R. Jones, Matthias Schonlau, and William J. Welch. Efficient global optimization of expensive black-box functions. *Journal of Global Optimization*, 13(4):455–492, 1998.
- [32] Kirtveasan Kandasamy, Jeff Schneider, and Barnabas Poczos. High dimensional bayesian optimisation and bandits via additive models. In *International Conference on Machine Learning (ICML)*, 2015.
- [33] Jack PC Kleijnen, Wim van Beers, and Inneke Van Nieuwenhuijse. Expected improvement in efficient global optimization through bootstrapped kriging. *Journal of Global Optimization*, 54(1):59–73, 2012.
- [34] Rogier Koppejan and Shimon Whiteson. Neuroevolutionary reinforcement learning for generalized control of simulated helicopters. *Evolutionary intelligence*, 4(4):219–241, 2011.
- [35] Andreas Krause and Carlos Guestrin. Nonmyopic active learning of Gaussian processes: an exploration-exploitation approach. In *International Conference on Machine Learning (ICML)*, Corvallis, Oregon, June 2007.
- [36] Oliver Kroemer, Renaud Detry, Justus Piater, and Jan Peters. Combining active learning and reactive control for robot grasping. *Robotics and Autonomous Systems*, 58(9):1105–1116, 2010.
- [37] Harold J. Kushner. A new method of locating the maximum of an arbitrary multipeak curve in the presence of noise. *Journal of Basic Engineering*, 86:97–106, 1964.
- [38] John F Lazenby. *The defence of Greece: 490-479 BC*. Aris & Phillips, 1993.
- [39] Nimalan Mahendran, Ziyu Wang, Firas Hamze, and Nando de Freitas. Adaptive MCMC with Bayesian optimization. *Journal of Machine Learning Research - Proceedings Track for Artificial Intelligence and Statistics (AISTATS)*, 22:751–760, 2012.
- [40] Ruben Martinez-Cantin. BayesOpt: A Bayesian optimization library for nonlinear optimization, experimental design and bandits. *Journal of Machine Learning Research*, 15:3735–3739, 2014.
- [41] Ruben Martinez-Cantin, Nando de Freitas, Eric Brochu, Jose Castellanos, and Arnaud Doucet. A Bayesian exploration-exploitation approach for optimal online sensing and planning with a visually guided mobile robot. *Autonomous Robots*, 27(3):93–103, 2009.
- [42] Ruben Martinez-Cantin, Nando de Freitas, Arnaud Doucet, and Jose A. Castellanos. Active policy learning for robot planning and exploration under uncertainty. In *Robotics: Science and Systems*, 2007.

- [43] Jonas Mockus. *Bayesian Approach to Global Optimization*, volume 37 of *Mathematics and Its Applications*. Kluwer Academic Publishers, 1989.
- [44] Jonas Mockus, Vytautas Tiesis, and Antanas Zilinskas. The application of Bayesian methods for seeking the extremum. In L.C.W. Dixon and G.P. Szego, editors, *Towards Global Optimisation 2*, pages 117–129. Elsevier, 1978.
- [45] Andrew W. Moore and Jeff Schneider. Memory-based stochastic optimization. In David S. Touretzky, Michael C. Mozer, and Michael E. Hasselmo, editors, *Advances in Neural Information Processing Systems*, volume 8, pages 1066–1072. The MIT Press, 1996.
- [46] Anthony O’Hagan. Some Bayesian numerical analysis. *Bayesian Statistics*, 4:345–363, 1992.
- [47] Carl E. Rasmussen and Christopher K.I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, Cambridge, Massachusetts, 2006.
- [48] Daniel Russo and Benjamin Van Roy. Learning to optimize via posterior sampling. *Mathematics of Operations Research*, 39(4):1221–1243, 2014.
- [49] Paul D Sampson and Peter Guttorp. Nonparametric estimation of nonstationary spatial covariance structure. *Journal of the American Statistical Association*, 87(417):108–119, 1992.
- [50] Amar Shah, Andrew Gordon Wilson, and Zoubin Ghahramani. Student-t processes as alternatives to Gaussian processes. In *AISTATS, JMLR Proceedings. JMLR.org*, 2014.
- [51] Hanif D Sherali and Vikram Ganesan. A pseudo-global optimization approach with application to the design of containerships. *Journal of Global Optimization*, 26(4):335–360, 2003.
- [52] Jasper Snoek, Hugo Larochelle, and Ryan Adams. Practical Bayesian optimization of machine learning algorithms. In *NIPS*, pages 2960–2968, 2012.
- [53] Jasper Snoek, Oren Rippel, Kevin Swersky, Ryan Kiros, Nadathur Satish, Narayanan Sundaram, Md. Mostofa Ali Patwary, Prabhat, and Ryan P. Adams. Scalable bayesian optimization using deep neural networks. In *International Conference on Machine Learning*, 2015.
- [54] Jasper Snoek, Kevin Swersky, Richard S. Zemel, and Ryan Prescott Adams. Input warping for Bayesian optimization of non-stationary functions. In *International Conference on Machine Learning*, 2014.
- [55] Niranjan Srinivas, Andreas Krause, Sham Kakade, and Matthias Seeger. Gaussian process optimization in the bandit setting: No regret and experimental design. In *Proc. International Conference on Machine Learning (ICML)*, 2010.
- [56] Bruce E Stuckman and Eric E Easom. A comparison of bayesian/sampling global optimization techniques. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(5):1024–1032, 1992.
- [57] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1998.
- [58] Kevin Swersky, David Duvenaud, Jasper Snoek, Frank Hutter, and Michael A Osborne. Raiders of the lost architecture: Kernels for bayesian optimization in conditional parameter spaces. In *NIPS workshop on Bayesian Optimization*, arXiv preprint arXiv:1409.4011, 2014.
- [59] Matthew A Taddy, Herbert KH Lee, Genetha A Gray, and Joshua D Griffin. Bayesian guided pattern search for robust local optimization. *Technometrics*, 51(4):389–401, 2009.
- [60] Marc Toussaint. The Bayesian search game. In Alberto Moraglio Yossi Borenstein, editor, *Theory and Principled Methods for Designing Metaheuristics*. Springer, 2014.
- [61] Doniyor Ulmasov, Caroline Baroukh, Benoit Chachuat, Marc Peter Deisenroth, and Ruth Misener. Bayesian optimization with dimension scheduling: Application to biological systems. In *26th European Symposium on Computer Aided Process Engineering, Computer-Aided Chemical Engineering*, 2016.

- [62] Ziyu Wang, Masrour Zoghi, Frank Hutter, David Matheson, and Nando de Freitas. Bayesian optimization in high dimensions via random embeddings. In *International Joint Conferences on Artificial Intelligence (IJCAI) - Extended version: <http://arxiv.org/abs/1301.1942>*, 2013.
- [63] Fernando Wario, Benjamin Wild, Margaret Jane Couvillon, Raúl Rojas, and Tim Landgraf. Automatic methods for long-term tracking and the detection and decoding of communication dances in honeybees. *Frontiers in Ecology and Evolution*, 3:103, 2015.
- [64] Eric R Westervelt, Jessy W Grizzle, Christine Chevallereau, Jun Ho Choi, and Benjamin Morris. *Feedback control of dynamic bipedal robot locomotion*, volume 28. CRC press, 2007.
- [65] Brian J. Williams, Thomas J. Santner, and William I. Notz. Sequential design of computer experiments to minimize integrated response functions. *Statistica Sinica*, 10(4):1133–1152, 2000.
- [66] Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3-4):229–256, 1992.